CS 205 Artificial Intelligence Instructor: Dr. Eamonn Keogh Project 2: Feature Selection with Nearest Neighbor June 15, 2023	
Student1 Name: Nishat Ara Tania SID: 862394061 Email: ntani005@ucr.edu Contribution: 1. Implementation: KNN class, Forward Selection, Figure drawing, Data Processing, and Main functions 2. Report writing	Student2 Name: Shahriar M Sakib SID: 862393922 Email: ssaki004@ucr.edu Contribution: 1. Implementation: Accuracy, Cross Validation, Backward Selection, and Applying Feature Selection functions 2. Report writing

In completing this Project, I consulted:

1. https://www.dropbox.com/sh/ftzvcnntl2j5eiu/AAASovmq047jOCIaHjL4h4GIa?dl=0 (Slides of Dr. Eamonn Keogh)
2. Pandas documentation (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)
3. Diabetes Dataset (https://www.kaggle.com/datasets/mathchi/diabetes-data-set)

All content in the report is original. We have consulted none. We have acknowledged the websites where we visited.

Outline of the report:

- GitHub Repository:https://github.com/nishataratania/feature_selection_knn

Introduction: In this project, we have implemented the Nearest Neighbor algorithm with two Feature Selection Search Algorithms: Forward Search and Backward Search. We evaluate our implementation on three different size dataset. We have also applied our feature selection approaches on a public diabetes prediction dataset collected from Kaggle.

Feature Selection: Feature Selection is the process of finding the most important features in a dataset. There are many feature selection algorithms. We have applied two feature selection algorithms here in this project.
1. Forward Selection
2. Backward Elimination

We have applied these feature selection algorithms on three given datasets. We have also applied this on a public dataset from UCI archive named as diabetes.

Forward Selection: Forward feature selection is an iterative approach to select important features from a set of features. We start from 0 features and iteratively try all features to select the best set of features. Iteratively we keep adding the best features until we do not find any features that improve the model.

Backward Selection: Backward feature selection is also an iterative process where we start with all the features. Then we keep removing features one by one and evaluate the performance of the model and remove the least important feature which improves the model. This way we keep improving the model.

kNN: We have implemented a Nearest Neighbor Class with K as a parameter. It stores the training data and predicts labels of the unseen data based on the majority vote.

kFold Cross Validation: We have implemented kFold Cross validation approach to evaluate the Forward and Backward Search Feature Selection with kNN.

Dataset: We are given three different sizes of dataset to evaluate the feature selection approaches. We select datasets based on the selection strategy based on the birthdate of teammates.

1. Small dataset: **CS170_small_Data__18** is a dataset with 1000 instances with 10 features.
2. Large dataset: **CS170_large_Data__31** is a dataset with 2000 instances with 20 features.
3. XLarge dataset: **CS170_XXXlarge_Data__12** is a dataset with 4000 instances with 80 features.

Diabetes dataset: This is a public diabete dataset from Kaggle with 768 instances with 8 features. It is originally from the National Institute of Diabetes and Digestive and Kidney

Diseases. It is a binary class problem to predict whether a patient has diabetes based on 8 features described below.

1. Pregnancies: Number of times pregnant
2. Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. BloodPressure: Diastolic blood pressure (mm Hg)
4. SkinThickness: Triceps skin fold thickness (mm)
5. Insulin: 2-Hour serum insulin (mu U/ml)
6. BMI: Body mass index (weight in kg/(height in m)^2)
7. DiabetesPedigreeFunction: Diabetes pedigree function
8. Age: Age (years)

Data Normalization: As this dataset has continuous valued features. We have normalized the dataset using min-max normalization. We followed the function to normalize the dataset,
normalized_dataset = (data - min_val) / (max_val - min_val)

Result:

Dataset	Forward Selection			Backward Selection		
	Selected Features	10-Fold Cross Validation Accuracy	Processing Time	Selected Features	10-Fold Cross Validation Accuracy	Processing Time
Small Dataset	4, 10	0.965	66.6 seconds	4, 10	0.965	89.6 seconds
Large Dataset	10, 20	0.977	26 minutes 27 seconds	4, 5, 20	0.856	40 minutes 56 seconds
XLarge Dataset	10, 63, 71	0.97	107 hours 21 minutes 34 seconds	-	-	
Diabetes Dataset	Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age	0.75	22.68 Seconds	Pregnancies, Glucose, Age	0.75	29.4 Seconds

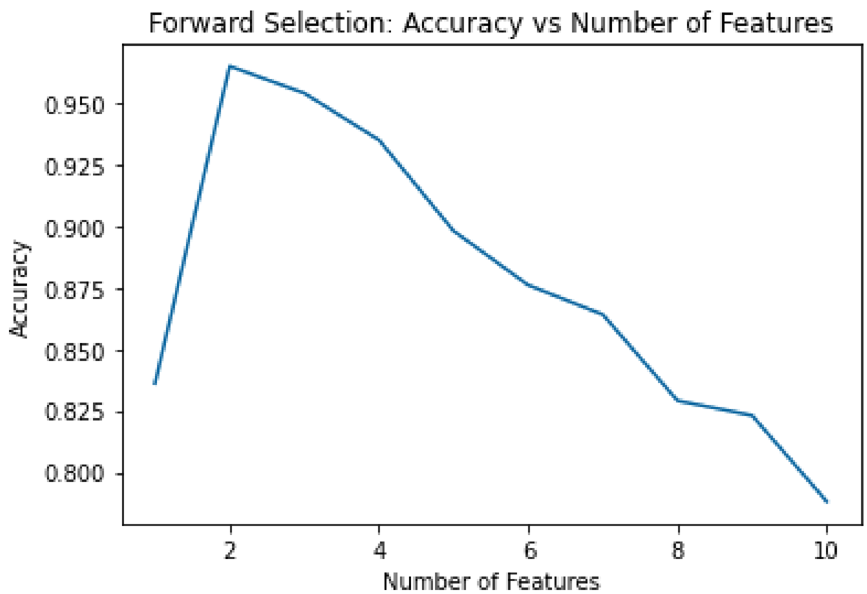


Figure 1: Forward feature selection performance on **CS170_small_Data__18**

We can see from Figure 1 that using only feature 4, it shows 83.6% accuracy. Next after adding features 10, the accuracy goes up to 97.2% accuracy. It does not improve the accuracy if we add more features. As a result, it selects the combination of features 4 and 10 to give the highest accuracy 97.2%.

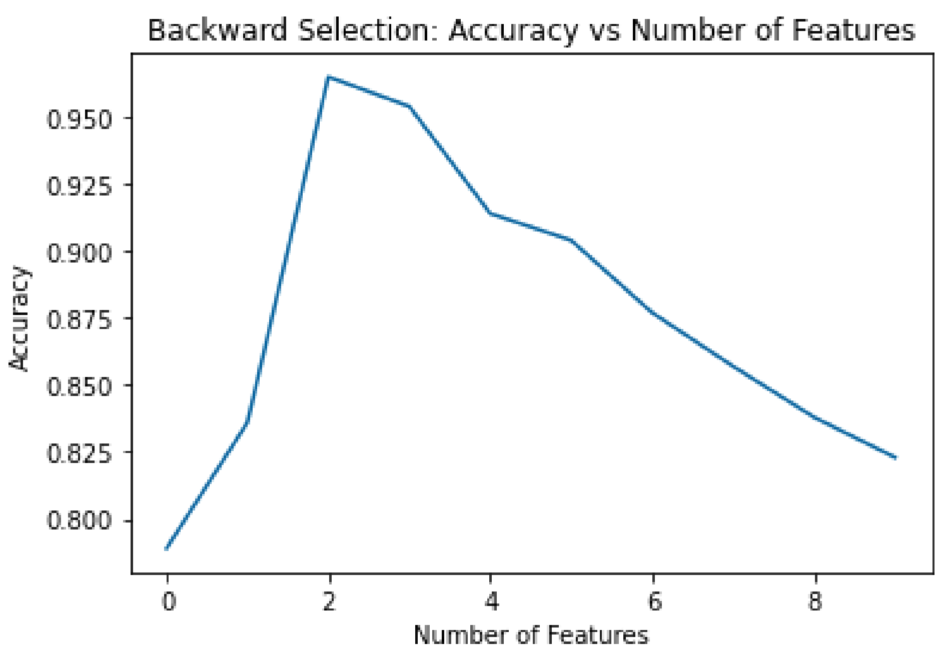


Figure 2: Backward feature selection performance on **CS170_small_Data__18**

From Figure 2, we can see that Backward Search on the small dataset selects the same combination of features: 4 and 10 as the best performance features with 96.5% accuracy. It first starts with 78.4% accuracy with all the features. In the later stages, it sequentially eliminates the worst performing features. Eventually, it keeps only the most important set of features.

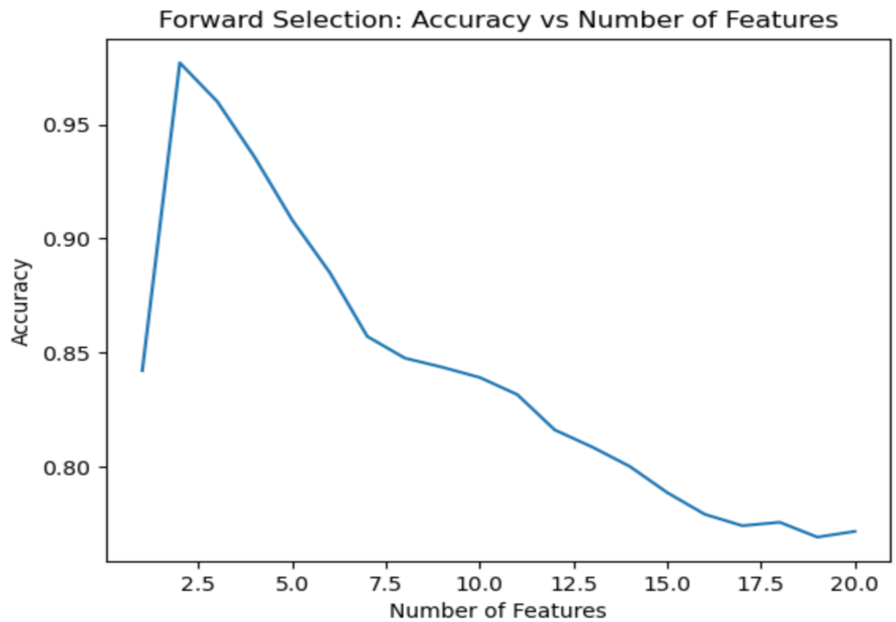


Figure 3: Forward feature selection on **CS170_large_Data__31**

We plot the features and accuracy of every level of forward feature search on the large dataset in Figure 3. It keeps selecting the best performing features in every level. It starts with selecting feature 20 with accuracy 85.8%. Eventually it selects the best combination of features: 10, and 20 with accuracy 97.7%.

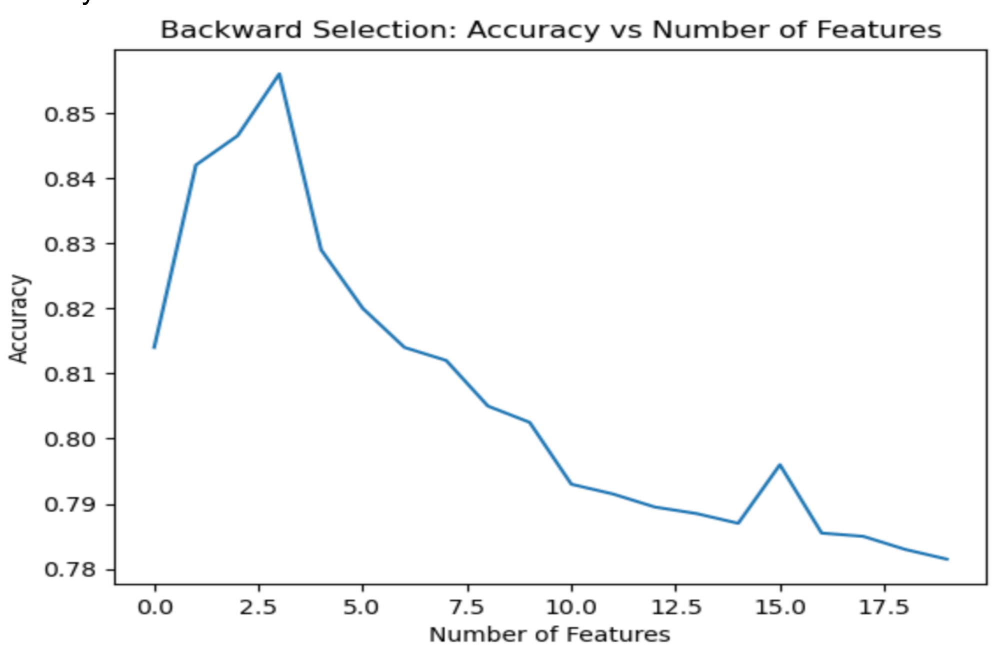


Figure 4: Backward feature selection on **CS170_large_Data__31**

From Figure 4, we can see that backward feature selection selects 4, 5, and 20 with 85.6% accuracy while eliminating worst performing features one by one. Interestingly, its performance is worse than forward selection approaches.

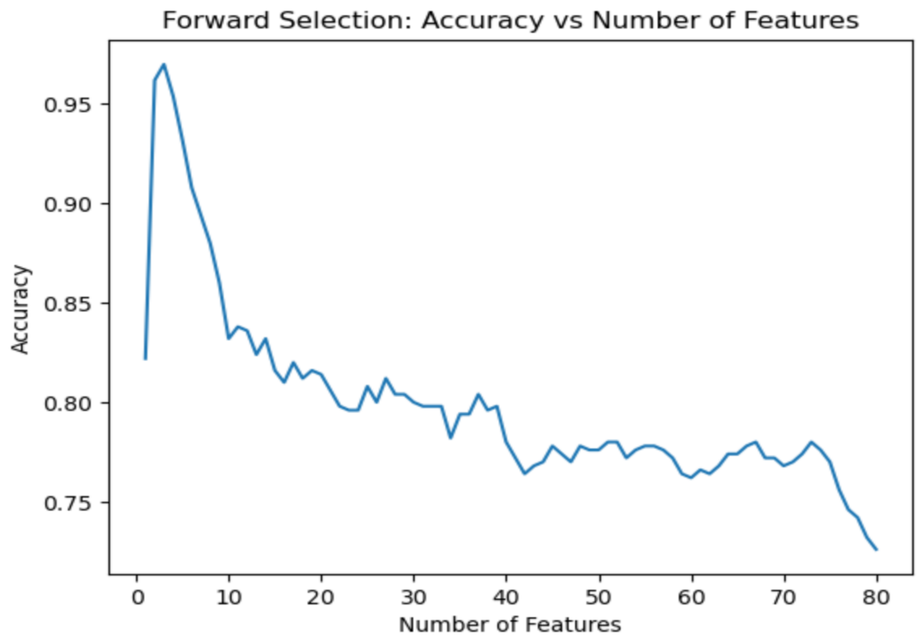


Figure 5: Forward features selection on Dataset: **CS170_XXXlarge_Data__12**

Since this is a big dataset, we apply Forward Search on 50% samples of the given extra large dataset. Cross validation accuracy of KNN vs Number of features have been plotted in Figure 5. We can see that it selects only 3 features: 10, 63, and 71 as the most important features with 97% accuracy.

Forward Selection on Diabetes Dataset: I have applied the Forward Feature Selection Algorithm on the Diabetes dataset. At first, it identified **Glucose** as the best informative feature and selected it in the first level with **67.48%** accuracy which is intuitive as we know glucose level quantifies the diabetes level. So, obviously Glucose will be the most important feature predicting diabetes for a person. Later, it selects DiabetesPedigreeFunction with the Glucose as the best combination to predict diabetes of a person. This way, it keeps searching and selects the combination of *Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigree Function, and Age* as the best features with 75.01% accuracy.

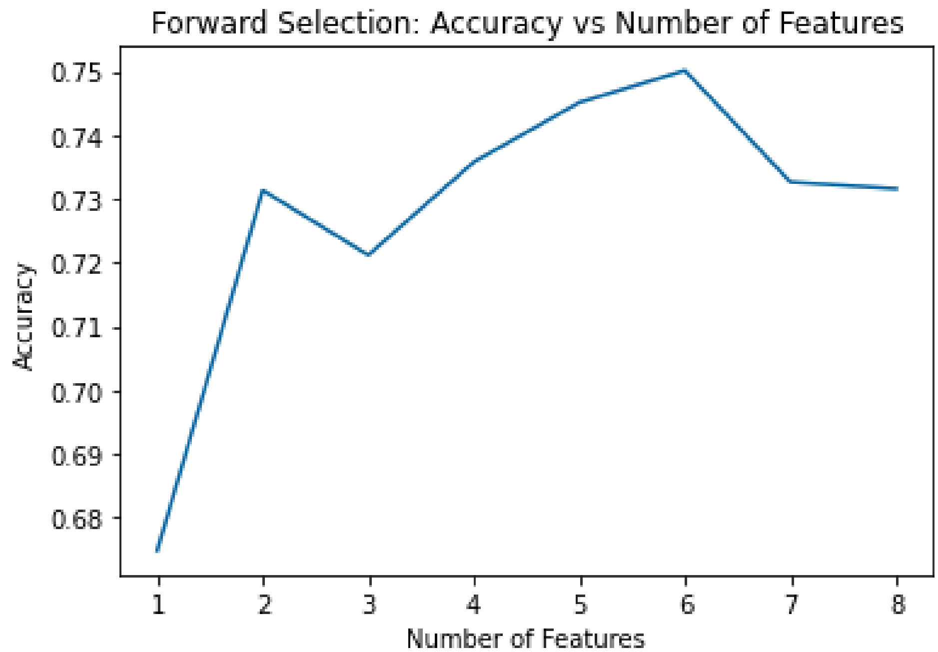


Figure 6: Forward search on diabete dataset selects: Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, and Age as the best features

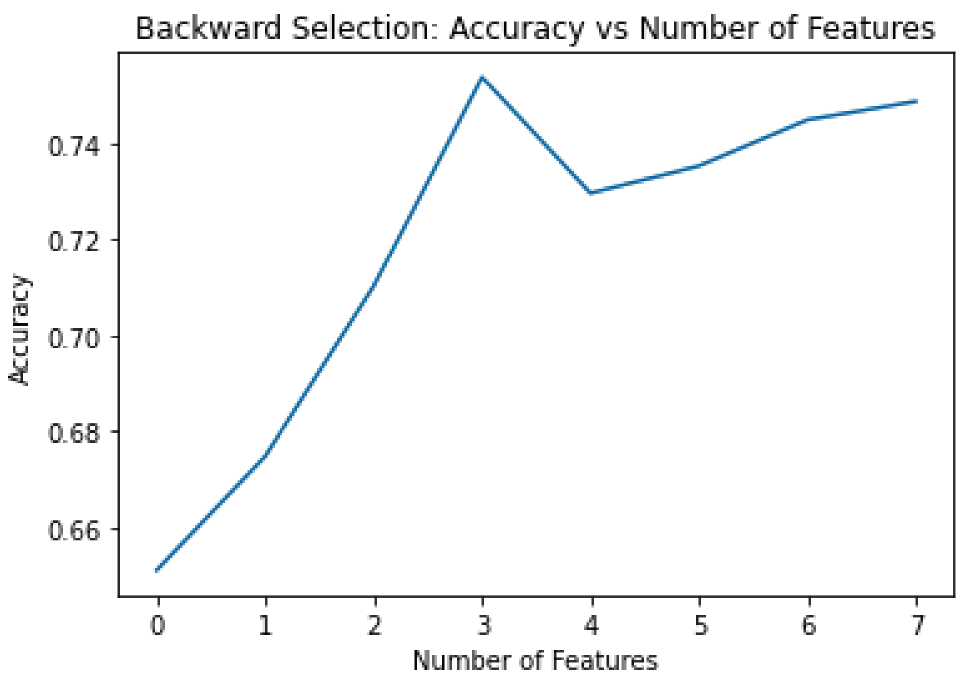


Figure 7: Backward Feature Selection Search selects features: Pregnancies, Glucose, and Age with 75.4% accuracy

Backward Selection on Diabetes Dataset: I have applied the Backward Feature Selection Algorithm on the Diabetes dataset. Initially, it shows 73.16% accuracy using 10-Fold cross validation in detecting diabetes using all features in the dataset. Then, it starts removing the worst performing features. At first, it removed **SkinThickness** as the least informative feature and removed it in the first level with **74.87%** accuracy. It makes sense as skin thickness does not have much correlation with having diabetes for a person. This way, it keeps searching and selects the combination of **Pregnancies, Glucose, and Age** as the best features with 75.04% accuracy.

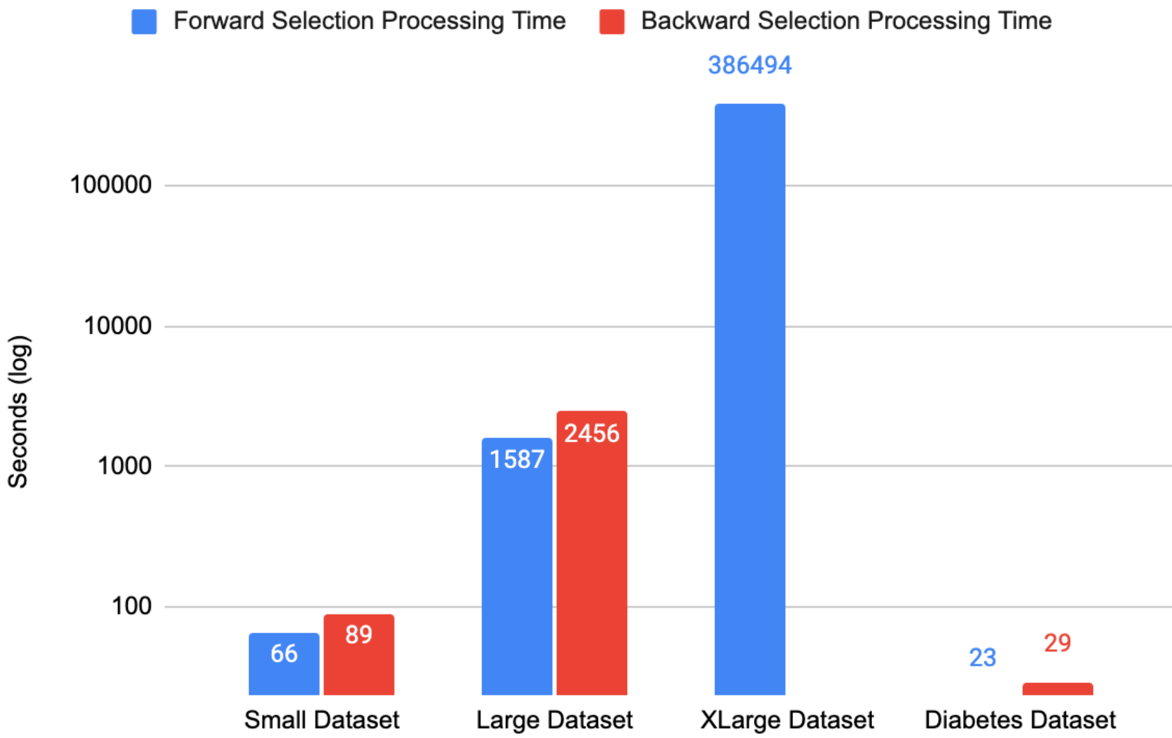


Figure 8: Processing time of Forward and Backward Selection on different datasets

Processing time: We have used a jupyter notebook in a 2.0GHz Quad Core Intel Core i5 Mac OS to test our implementation. We are showing the processing time for cross validation of different feature selection algorithms on different dataset in Figure 8.

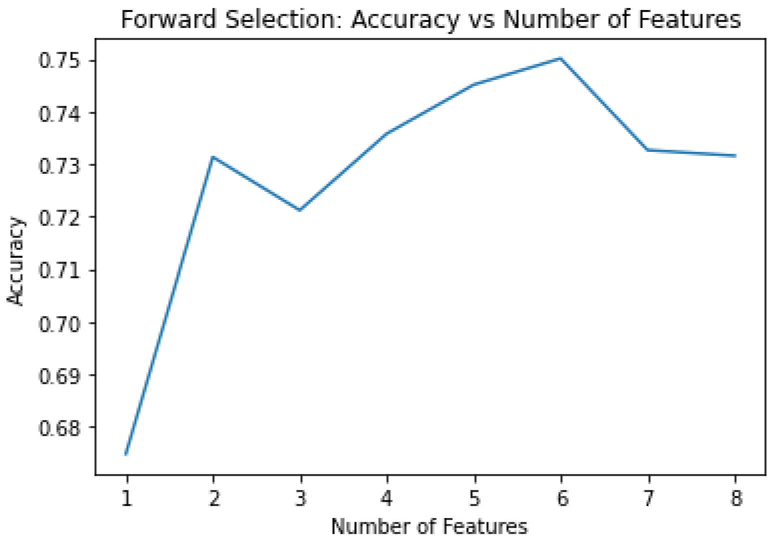
Conclusion: We have applied two different feature selection algorithms - Forward Selection, and Backward Elimination Search algorithm on four different datasets and found out the important as well as unimportant features in those dataset. We have presented the intuition behind their importance and unimportance in the diabete dataset. We have used KNN and 10-fold cross validation for accuracy measurement.

Outputs:

```
Welcome to Feature Selection Algorithm!

Select the dataset you want to test
            1 for Small Dataset
            2 for Large Dataset
            3 for XXX_Large Dataset
            4 for Diabetes Dataset 4
Select feature selection option
            1 for Forward Feature Selection
            2 for Backward Feature Selection 1

(768, 9)
Starting Forward Selection Search

 0%|                                                  | 0/8 [00:00<?, ?it/s]
Level  1  using  1  features:
           Accuracy:  0.5721804511278197  after adding  1
           Accuracy:  0.6748120300751881  after adding  2
           Accuracy:  0.5973057644110276  after adding  3
           Accuracy:  0.581077694235589  after adding  4
           Accuracy:  0.6216791979949876  after adding  5
           Accuracy:  0.6379699248120301  after adding  6
           Accuracy:  0.5650375939849624  after adding  7

12%|                                            | 1/8 [00:38<04:28, 38.29s/it]
           Accuracy:  0.6216791979949875  after adding  8
Best feature:  2  with accuracy:  0.6748120300751881
Feature list:  [2]

Level  2  using  2  features:
           Accuracy:  0.7027568922305764  after adding  1
           Accuracy:  0.693922305764411  after adding  3
           Accuracy:  0.6739348370927318  after adding  4
           Accuracy:  0.6760025062656643  after adding  5
           Accuracy:  0.699436090225564  after adding  6
           Accuracy:  0.731328320802005  after adding  7

25%|                                            | 2/8 [01:27<04:27, 44.50s/it]
           Accuracy:  0.7102130325814537  after adding  8
Best feature:  7  with accuracy:  0.731328320802005
Feature list:  [2, 7]

Level  3  using  3  features:
           Accuracy:  0.7109022556390978  after adding  1
           Accuracy:  0.7015664160401003  after adding  3
           Accuracy:  0.7102756892230577  after adding  4
           Accuracy:  0.7114661654135339  after adding  5
           Accuracy:  0.7211779448621554  after adding  6

38%|                                            | 3/8 [02:19<04:01, 48.20s/it]
```

```
        Accuracy:  0.718671679197995  after adding  8
Attention!! Accuracy has not improved by adding this feature!
Best feature:  6  with accuracy:  0.7211779448621554
Feature list:  [2, 7, 6]

Level  4  using  4  features:
        Accuracy:  0.7223684210526315  after adding  1
        Accuracy:  0.7025062656641603  after adding  3
        Accuracy:  0.7117167919799499  after adding  4
        Accuracy:  0.7273809523809524  after adding  5

50%|█████████████████████                    | 4/8 [03:12<03:20, 50.05s/it]
        Accuracy:  0.7357769423558898  after adding  8
Best feature:  8  with accuracy:  0.7357769423558898
Feature list:  [2, 7, 6, 8]

Level  5  using  5  features:
        Accuracy:  0.7363408521303259  after adding  1
        Accuracy:  0.7347117794486216  after adding  3
        Accuracy:  0.731829573934837  after adding  4

62%|██████████████████████████               | 5/8 [04:05<02:32, 50.96s/it]
        Accuracy:  0.7451127819548873  after adding  5
Best feature:  5  with accuracy:  0.7451127819548873
Feature list:  [2, 7, 6, 8, 5]

Level  6  using  6  features:
        Accuracy:  0.7342105263157894  after adding  1
        Accuracy:  0.7238095238095238  after adding  3

75%|███████████████████████████████▌         | 6/8 [04:55<01:41, 50.73s/it]
        Accuracy:  0.750125313283208  after adding  4
Best feature:  4  with accuracy:  0.750125313283208
Feature list:  [2, 7, 6, 8, 5, 4]

Level  7  using  7  features:
        Accuracy:  0.7326441102756893  after adding  1

88%|████████████████████████████████████▊    | 7/8 [05:31<00:45,
45.80s/it]
        Accuracy:  0.7288220551378447  after adding  3
Attention!! Accuracy has not improved by adding this feature!
Best feature:  1  with accuracy:  0.7326441102756893
Feature list:  [2, 7, 6, 8, 5, 4, 1]

Level  8  using  8  features:

100%|██████████████████████████████████████████| 8/8 [05:50<00:00,
43.85s/it]
        Accuracy:  0.7315789473684211  after adding  3
Attention!! Accuracy has not improved by adding this feature!
Best feature:  3  with accuracy:  0.7315789473684211
```

```
Feature list:  [2, 7, 6, 8, 5, 4, 1, 3]

Selected features: [2, 4, 5, 6, 7, 8]
Best accuracy:  0.750125313283208
```




```
Total time taken: 351.12572479248047 seconds
Welcome to Feature Selection Algorithm!

Select the dataset you want to test
            1 for Small Dataset
            2 for Large Dataset
            3 for XXX_Large Dataset
            4 for Diabetes Dataset
```

Code:

```
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import operator
from tqdm import tqdm
import time
class KNN_Classifier:
    def __init__(self, K=1):
        self.n_neighbor = K

    def fit(self, X, Y):
        self.train_data = X
        self.train_labels = Y

    def get_distance(self, train_X, test_X):
```

```
        total = 0
        for i in range(len(train_X)):
            total = total + np.square(test_X[i]-train_X[i])
        return np.sqrt(total)

    def predict_labels(self, test_X):
        test_labels=[]
        for test_d in test_X:
            pred_label = self.predict_label(test_d)
            test_labels.append(pred_label)
        return test_labels

    def predict_label(self, test_x):
        distance_map = {}
        for i in range(len(self.train_data)):
            dist = self.get_distance(test_x, self.train_data[i])
            distance_map[i]=dist

        sorted_dist = sorted(distance_map.items(), key = operator.itemgetter(1))

        neighbors_list = []
        for i in range(self.n_neighbor):
            neighbors_list.append(sorted_dist[i][0])


        majority_vote = {}
        for i in range(len(neighbors_list)):
            label = self.train_labels[neighbors_list[i]]

            if label in majority_vote:
                majority_vote[label]+=1
            else:
                majority_vote[label]=1

        sorted_vote = sorted(majority_vote.items(), key=operator.itemgetter(1), reverse=True)
        pred_label = sorted_vote[0][0]
        return pred_label
def accuracy_score(actual_labels, pred_labels):
    correct = 0
    for i in range(len(actual_labels)):
        if actual_labels[i] == pred_labels[i]:
            correct += 1
    accuracy = correct/len(pred_labels)
    return accuracy

def cross_validation(model, data, target, k=10):
    data_size = data.shape[0]
    fold_size = data_size // k
    accuracies = []
```

```
    for i in range(k):
        start = i * fold_size
        if i < k-1:
            end = (i + 1) * fold_size
        else:
            end = data_size

        #select train data
        train_data = np.concatenate([data[:start], data[end:]])
        train_target = np.concatenate([target[:start], target[end:]])

        #select test data
        test_data = data[start:end]
        test_target = target[start:end]

        model.fit(train_data, train_target)
        predictions = model.predict_labels(test_data)

        accuracy = accuracy_score(test_target, predictions)
        accuracies.append(accuracy)
    # get avg accuracy of k accuracies
    accuracy = np.mean(accuracies)
    return accuracy
def get_data_for_selected_features(df, selected_features):
    X = df[selected_features].to_numpy()
    Y = df['label'].to_numpy()
    return X, Y

def forward_selection(df, accuracy_dict):
    features = get_features_list(df)
    selected_features = []
    K = 3
    all_best_accuracy = 0.0
    all_best_features = []
    num_features = []  # list to keep track of the number of features
    accuracies = []  # list to keep track of the accuracies
    print("Starting Forward Selection Search")
    prev_acc = 0.0
    for i in tqdm(range(len(features))):
        print("Level ", i+1, " using ", i+1, " features: ")
        best_accuracy = 0.0
        best_feature = None
        for feature in features:
            if feature not in selected_features:
                current_features = selected_features + [feature]
                current_features = sorted(current_features)
                key = "_".join(map(str, current_features))
                if key not in accuracy_dict:
                    X, Y = get_data_for_selected_features(df, current_features)
```

```
                model = KNN_Classifier(K=K)
                accuracy = cross_validation(model, X, Y)
                accuracy_dict[key] = accuracy
            else:
                accuracy = accuracy_dict[key]
            if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_feature = feature

            print("\tAccuracy: ", accuracy, " after adding ", feature)

        if best_feature is not None:  # Only add feature if one improved the model
            selected_features.append(best_feature)
            features.remove(best_feature)  # Remove the selected feature from the original list
            num_features.append(len(selected_features))  # append the current number of
features
            accuracies.append(best_accuracy)  # append the current best accuracy

        if best_accuracy > all_best_accuracy:
            all_best_accuracy = best_accuracy
            all_best_features = selected_features.copy()
            print("Best feature: ", best_feature, " with accuracy: ", best_accuracy)
            print("Feature list: ", selected_features)
        else:
            print("Attention!! Accuracy has not improved by adding this feature!")
            print("Best feature: ", best_feature, " with accuracy: ", best_accuracy)
            print("Feature list: ", selected_features)
        print()

    all_best_features = sorted(all_best_features)
    print(f'Selected features: {all_best_features}')
    print("Best accuracy: ", all_best_accuracy)

    return num_features, accuracies, all_best_features, accuracy_dict
def backward_selection(df, accuracy_dict):
    features = get_features_list(df)
    selected_features = features  # start with all features
    K = 3
    all_best_accuracy = 0.0
    all_best_features = selected_features.copy()

    # First evaluate the model with all features
    X, Y = get_data_for_selected_features(df, selected_features)

    model = KNN_Classifier(K=K)
    accuracy = cross_validation(model, X, Y)
    key = "_".join(map(str, selected_features))
    accuracy_dict[key] = accuracy
    all_best_accuracy = accuracy
    print("Initial accuracy with all features: ", all_best_accuracy)
```

```
    num_features = []  # list to keep track of the number of features
    accuracies = []  # list to keep track of the accuracies
    print("Starting Backward Selection Search")
    i = len(features)
    while i > 0:
        print("Level ", i, " using ", i, " features: ")
        best_accuracy = 0.0
        best_feature = None
        for feature in selected_features:
            current_features = selected_features.copy()
            current_features.remove(feature)
            current_features = sorted(current_features)
            key = "_".join(map(str, current_features))
            if key not in accuracy_dict:
                X, Y = get_data_for_selected_features(df, current_features)

                model = KNN_Classifier(K=K)
                accuracy = cross_validation(model, X, Y)
                accuracy_dict[key] = accuracy
            else:
                accuracy = accuracy_dict[key]

            if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_feature = feature

            print("\tAccuracy: ", accuracy, " after removing ", feature)

        if best_feature is not None:  # Only remove feature if it improved the model
            selected_features.remove(best_feature)  # Remove the selected feature from the
original list
            num_features.append(len(selected_features))  # append the current number of
features
            accuracies.append(best_accuracy)  # append the current best accuracy
            i -= 1

        if best_accuracy >= all_best_accuracy:
            all_best_accuracy = best_accuracy
            all_best_features = selected_features.copy()
            print("Removed feature: ", best_feature, " with accuracy: ", best_accuracy)
            print("Current feature list: ", selected_features)
        else:
            print("Attention!! Accuracy has not improved by removing this feature!")
            print("Removed feature: ", best_feature, " with accuracy: ", best_accuracy)
            print("Current feature list: ", selected_features)
        print()

    all_best_features = sorted(all_best_features)
    print(f'Selected features: {all_best_features}')
```

```
    print("Best accuracy: ", all_best_accuracy)

    return num_features, accuracies, all_best_features, accuracy_dict

def load_diabetes_dataset(filename = 'diabetes.csv'):
    data = pd.read_csv(filename)

    # Save original column names
    original_columns = data.columns.tolist()

    # Prepare new column names
    new_cols = ['label' if col=='label' else i+1 for i, col in enumerate(data.columns)]

    # Rename columns
    data.columns = new_cols

    # Create a dictionary mapping new column names to old ones
    col_name_mapping = {new: old for new, old in zip(new_cols, original_columns)}

    return data, col_name_mapping

def min_max_scaler(data):
    min_val = data.min()
    max_val = data.max()

    data_normalized = (data - min_val) / (max_val - min_val)

    return data_normalized

def shuffle_normalize(df):
    df = df.sample(frac=1.0, random_state=42)

    normalized_df = df.copy()
    features = df.columns.drop('label')
    normalized_df[features] = min_max_scaler(df[features])

    return normalized_df

def get_features_list(data):
    columns = list(data.columns)
    features = []
    for val in columns:
        if val != 'label':
            features.append(val)
    return features

def plot_feature_count_vs_accuracies(num_features, accuracies, title):
    plt.plot(num_features, accuracies)
    plt.xlabel('Number of Features')
    plt.ylabel('Accuracy')
```

```
    plt.title(title)
    plt.show()

def load_dataset(file_name="CS170_small_Data__18.txt"):
    if file_name == "diabetes.csv":
        return load_diabetes_dataset()
    # Determine the number of columns in your data file if not known
    with open(file_name, 'r') as f:
        line = f.readline()
        num_cols = len(line.split())

    # Create column names
    col_names = ['label'] + [i+1 for i in range(num_cols - 1)]

    # Load the data
    data = pd.read_csv(file_name, names=col_names, delim_whitespace=True)

    if file_name == "CS170_XXXlarge_Data__12.txt":
        print("Sampling 50% of dataset!!")
        data = data.sample(frac=0.5, random_state=42).reset_index(drop=True)

    return data, col_names
def apply_feature_selection(feature_selection, file_name, accuracy_cache):
    data, _ = load_dataset(file_name)
    data = shuffle_normalize(data)
    print(data.shape)

    # test_frac=0.25
    # train_df, test_df = train_test_split(data, test_frac)


    if feature_selection == '1':
        num_features1, accuracies1, \
        all_best_features1, accuracy_cache = forward_selection(data,accuracy_cache)
        title = 'Forward Selection: Accuracy vs Number of Features'
        plot_feature_count_vs_accuracies(num_features1, accuracies1, title)

    elif feature_selection == '2':
        num_features1, accuracies1, \
        all_best_features1, accuracy_cache = backward_selection(data,accuracy_cache)
        title = 'Backward Selection: Accuracy vs Number of Features'
        plot_feature_count_vs_accuracies(num_features1, accuracies1, title)
    return num_features1, accuracies1, all_best_features1, accuracy_cache

if __name__ == "__main__":
    small_data_accuracy_cache = {}
    large_data_accuracy_cache = {}
    xlarge_data_accuracy_cache = {} #get_accuracy_dict()
    diabetes_data_accuracy_cache = {}
    while(True):
```

```
    print("Welcome to Feature Selection Algorithm!")
    data_selection = input("Select the dataset you want to test\n" + \
                           "\t 1 for Small Dataset \n" + \
                           "\t 2 for Large Dataset \n" + \
                           "\t 3 for XXX_Large Dataset \n" + \
                           "\t 4 for Diabetes Dataset")

    feature_selection = input("Select feature selection option \n" + \
                            "\t 1 for Forward Feature Selection \n" + \
                            "\t 2 for Backward Feature Selection")

    if data_selection == '1':
        start_time = time.time()
        file_name="data_sets/CS170_small_Data__18.txt"
        #data = load_dataset(file_name)
        num_features1, accuracies1, all_best_features1, large_data_accuracy_cache =
apply_feature_selection(
            feature_selection, file_name, large_data_accuracy_cache)
        end_time = time.time()
        print(f"Total time taken: {end_time - start_time} seconds")

    elif data_selection == '2':
        start_time = time.time()
        file_name="data_sets/CS170_large_Data__31.txt"
        num_features1, accuracies1, all_best_features1, large_data_accuracy_cache =
apply_feature_selection(
            feature_selection, file_name, large_data_accuracy_cache)
        end_time = time.time()
        print(f"Total time taken: {end_time - start_time} seconds")

    elif data_selection == '3':
        start_time = time.time()
        print(len(xlarge_data_accuracy_cache))
        file_name="data_sets/CS170_XXXlarge_Data__12.txt"
        num_features1, accuracies1, all_best_features1, xlarge_data_accuracy_cache =
apply_feature_selection(
            feature_selection, file_name, xlarge_data_accuracy_cache)
        end_time = time.time()
        print(f"Total time taken: {end_time - start_time} seconds")

    elif data_selection == '4':
        start_time = time.time()
        file_name = 'diabetes.csv'
        num_features1, accuracies1, all_best_features1, diabetes_data_accuracy_cache =
apply_feature_selection(
            feature_selection, file_name, diabetes_data_accuracy_cache)
        end_time = time.time()
        print(f"Total time taken: {end_time - start_time} seconds")
```